Support Classes

Objectives

- Support Classes: Collections, Arrays
- Use the Collections class
  - Sorting/ Shuffling
  - Routine Data Manipulation(copy, reverse, swap, addAll…)
  - Searching (binarySearch)/ Composition(frequency, disjoint, min, max)
  - Finding Extreme Values (to find min,max value by comparator)
- Use the Arrays class

Introduction

- java.lang.**Object**
  - java.util.**Arrays**
  - java.util.**Collections**

- An algorithm on a list can be applied on some lists although the type of elements in each list can be different.
- The *polymorphic algorithms* described here are pieces of reusable functionality provided by the Java platform.
- All of them come from the **Collections** class and the **Arrays** class (support classes), and all take the form of static methods whose first argument is the collection on which the operation is to be performed.

The Collections class

- A support class containing static methods which accept collections as their parameters.
- file:///J:/Softs/JavaSofts/JavaDocs/docs-Java8/api/java/util/Collections.html

Routine Data Manipulation

- The Collections class provides five algorithms for doing routine data manipulation on List objects, including:
  - reverse()
  - fill()
  - copy()
  - swap()
  - addAll()

Searching

- Condition: The list in ascending order
- The binarySearch algorithm searches for a specified element in a sorted List.
  - Return pos >=0 → Present
  - Return pos<0 → Absent

Composition

- `frequency` — counts the number of times the specified element occurs in the specified collection.
- `disjoint` — determines whether two Collections are disjoint; that is, whether they contain no elements in common.

Finding Extreme Values

- Methods: min(…), max(…)

Collections Demo.

```
import java.util.ArrayList;
import java.util.Vector;
import java.util.Collections;
import java.util.Random;
public class CollectionsDemo {
    public static void main(String[] args){
        ArrayList ar= new ArrayList();
        Vector v = new Vector();
        Random rd= new Random(); // MAXIMUM VALUE= 29
        for (int i=1; i<=10; i++){
            ar.add(rd.nextInt(30));
            v.add(rd.nextInt(30));
        }
        System.out.println("ar=" + ar);
        System.out.println("v=" + v);
        boolean dis= Collections.disjoint(ar, v);
        System.out.println("ar and v are disjunct: " + dis);
        Collections.addAll(v, ar.toArray());
        System.out.println("After adding, v=" + v);
        int minVal= (int)Collections.min(v);
        int maxVal= (int) Collections.max(v);
```

Collections Demo.

```
        System.out.println("min= " + minVal + ", max= "+ maxVal);
        int fre= Collections.frequency(v, 8);
        System.out.println("Occurences of 8: " + fre);
        Collections.sort(v);
        System.out.println("After sorting, v=" + v);
        int pos = Collections.binarySearch(v, 8);
        System.out.println("Position of 8: " + pos);
        Collections.shuffle(v);
        System.out.println("After shuffling, v=" + v);

    }
}
```

```
run:
ar=[16, 22, 13, 29, 12, 8, 23, 8, 17, 10]
v=[3, 2, 24, 13, 24, 18, 22, 8, 3, 1]
ar and v are disjunct: false
After adding, v=[3, 2, 24, 13, 24, 18, 22, 8, 3, 1, 16, 22, 13, 29, 12, 8, 23, 8, 17, 10]
min= 1, max= 29
Occurences of 8: 3
After sorting, v=[1, 2, 3, 3, 8, 8, 8, 10, 12, 13, 13, 16, 17, 18, 22, 22, 23, 24, 24, 29]
Position of 8: 4
After shuffling, v=[3, 3, 17, 8, 23, 8, 12, 24, 13, 18, 2, 24, 1, 29, 22, 16, 22, 13, 10, 8]
```

Sorting

- The sort algorithm reorders a List so that its elements are in ascending order according to an ordering relationship.
- Example

```
public class Sort {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

Sorting

```java
import java.util.ArrayList;
import java.util.Collections;

public class DemoSortString {
    public static void main(String[] args) {
        ArrayList list=new ArrayList();
        list.add("Giang");
        list.add("Hoang");
        list.add("Tuan");
        list.add("An");
        list.add("Binh");
        Collections.sort(list);

        System.out.println("List after sort:" + list);
    }
}
```

Output - JavaApplication22 (run)

```
run:
List after sort:[An, Binh, Giang, Hoang, Tuan]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Comparable Interface

- A comparison function, which imposes a total ordering on some collection of objects
- The following demonstration will show you the way to sort a list based on your own criteria: A list of employees will be sorted based on ascending IDs.

Comparable Interface

Employee.java

Source History

```
1
2   package DemoSort;
3
4   public class Employee implements Comparable {
5
6       String ID;
7       String Name;
8       int salary;
9
10      public Employee(String ID, String Name, int salary) {...5 lines }
15
16      public String getID() {...3 lines }
19
20      public void setID(String ID) {...3 lines }
23
24      public String getName() {...3 lines }
27
28      public void setName(String Name) {...3 lines }
31
32      public int getSalary() {...3 lines }
35
36      public void setSalary(int salary) {...3 lines }
39
        public String toString(){
41          return ID+"-"+ Name+"-"+ salary;
42      }
43
44      @Override
        public int compareTo(Object t) {
46          Employee emp=(Employee)t;
47          return ID.compareTo(emp.getID());// Ham so sanh chuoi
48      }
49  }
```

Comparable Interface

Employee.java | UsingEployee.java

Source | History

```java
package DemoSort;

import java.util.ArrayList;
import java.util.Collections;

public class UsingEployee {
    public static void main(String[] args) {
        ArrayList listemp=new ArrayList();
        Employee e1=new Employee("SE123", "Nguyen Hoai Bao", 5000);
        Employee e2=new Employee("SE129", "Nguyen Tien Linh", 4000);
        Employee e3=new Employee("SE120", "Tran Minh Vuong", 3000);

        listemp.add(e1);
        listemp.add(e2);
        listemp.add(e3);

        Collections.sort(listemp);

        System.out.println("Employee List :" + listemp);

    }

}
```

Output - JavaApplication22 (run)

```
run:
Employee List :[SE120-Tran Minh Vuong-3000, SE123-Nguyen Hoai Bao-5000, SE129-Nguyen Tien Linh-4000]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Comparator Interface

- A comparison function, which imposes a total ordering on some collection of objects
- The following demonstration will show you the way to sort a list based on your own criteria: A list of employees will be sorted based on descending salaries then ascending IDs.

Comparator Interface – Demo.

Employee.java | SortSalary.java | UsingEployee02.java

Source | History

```
1
2   package DemoSort02;
3
4
5   public class Employee{
6
7       String ID;
8       String Name;
9       int salary;
10
11      public Employee(String ID, String Name, int salary) {
12          this.ID = ID;
13          this.Name = Name;
14          this.salary = salary;
15      }
16
17      public String getID() {...3 lines }
20
21      public void setID(String ID) {...3 lines }
24
25      public String getName() {...3 lines }
28
29      public void setName(String Name) {...3 lines }
32
33      public int getSalary() {...3 lines }
36
37      public void setSalary(int salary) {...3 lines }
40
      public String toString(){
42          return ID+"-"+ Name+"-"+ salary;
43      }
44
45  }
```

Comparator Interface – Demo.

Employee.java | SortSalary.java | UsingEployee02.java

Source | History

```java

package DemoSort02;

import java.util.Comparator;


public class SortSalary  implements Comparator{

    @Override
    public int compare(Object t, Object t1) {

        Employee e1=(Employee)t;
        Employee e2=(Employee)t1;

        int d= e1.getSalary()-e2.getSalary();

        if(d>0) return -1;// Sap giam dan theo Salary
        if(d==0) return e1.getID().compareTo(e2.getID());// Sap tang dan theo ID
        return 1;

    }

}
```

Comparator Interface – Demo.

Employee.java | SortSalary.java | UsingEployee02.java

Source | History

```java
package DemoSort02;

import DemoSort.*;
import java.util.ArrayList;
import java.util.Collections;

public class UsingEployee02 {
    public static void main(String[] args) {
        ArrayList listemp=new ArrayList();
        Employee e1=new Employee("SE123", "Nguyen Hoai Bao", 5000);
        Employee e2=new Employee("SE129", "Nguyen Tien Linh", 4000);
        Employee e3=new Employee("SE120", "Tran Minh Vuong", 3000);
        Employee e4=new Employee("SE115", "Nguyen Van Toan", 4000);

        listemp.add(e1);
        listemp.add(e2);
        listemp.add(e3);
        listemp.add(e4);

       Collections.sort(listemp,new SortSalary());

       System.out.println("Employee List :");
       for(int i=0;i<listemp.size();i++)
            System.out.println(listemp.get(i));

    }

}
```

Comparator Interface – Demo.

Output - JavaApplication22 (run)

```
run:
Employee List :
SE123-Nguyen Hoai Bao-5000
SE115-Nguyen Van Toan-4000
SE129-Nguyen Tien Linh-4000
SE120-Tran Minh Vuong-3000
BUILD SUCCESSFUL (total time: 0 seconds)
```

The Arrays Class

- It it similar to the Collections class, but it accepts arrays as it's parameters.
- file:///J:/Softs/JavaSofts/JavaDocs/docs-Java8/api/java/util/Arrays.html

Arrays Class: Demo

```
run:
ar1=[5, 1, 4, 7, 9, 3, 4, 5, 3]
ar2=[5, 6, 7, 8, 9]
ar3=[5, 6, 7, 8, 9]
ar1=ar2: false
ar2=ar3: true
ar4= [5, 1, 4]
ar5=[4, 7, 9, 3]
After sorting, ar1=[1, 3, 3, 4, 4, 5, 5, 7, 9]
Binary search 7, pos= 7
```

```java
import java.util.Arrays;
public class ArraysDemo {
    public static void main(String[] args)
     int ar1[] = {5,1,4,7,9,3,4,5,3};
     int ar2[]= {5,6,7,8,9};
     int ar3[]= {5,6,7,8,9};
     System.out.println("ar1=" + Arrays.toString(ar1));
     System.out.println("ar2=" + Arrays.toString(ar2));
     System.out.println("ar3=" + Arrays.toString(ar3));
     boolean eq= Arrays.equals(ar1, ar2);
     System.out.println("ar1=ar2: " + eq);
     eq= Arrays.equals(ar2, ar3);
     System.out.println("ar2=ar3: " + eq);
     int numOfElements=3, from=2, before=6;
     int ar4[]= Arrays.copyOf(ar1, numOfElements);
     System.out.println("ar4= " + Arrays.toString(ar4));
     int ar5[] =Arrays.copyOfRange(ar1, from, before);
     System.out.println("ar5=" + Arrays.toString(ar5));
     Arrays.sort(ar1);
     System.out.println("After sorting, ar1=" + Arrays.toString(ar1));
     int pos = Arrays.binarySearch(ar1, 7);
     System.out.println("Binary search 7, pos= " + pos);
    }
}
```

Summary

- Support Classes: Collections, Arrays
- Use the Collections class
  - Sorting/ Shuffling
  - Routine Data Manipulation
  - Searching/ Composition
  - Finding Extreme Values
- Use the Arrays class